Tema 4: Estructuras de Control
Estructura y Contenidos

Tema 4: Estructuras de Control
OBJETIVOS

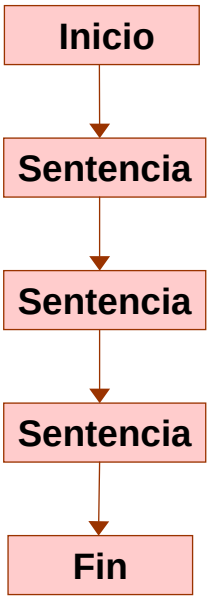
- ✓ Establecer la necesidad de las estructuras de control
- ✓ Diferenciar las estructuras de selección de las iterativas
- ✓ Estructuras de selección e iterativas en C
- ✓ Diseño de bucles: anidamientos y bucles infinitos
- ✓ Todo problema que se pueda resolver en un numero finito de pasos puede expresarse con el uso de estructuras secuenciales, selectivas e iterativas

Tema 4: Estructuras de Control
Estructura y Contenidos

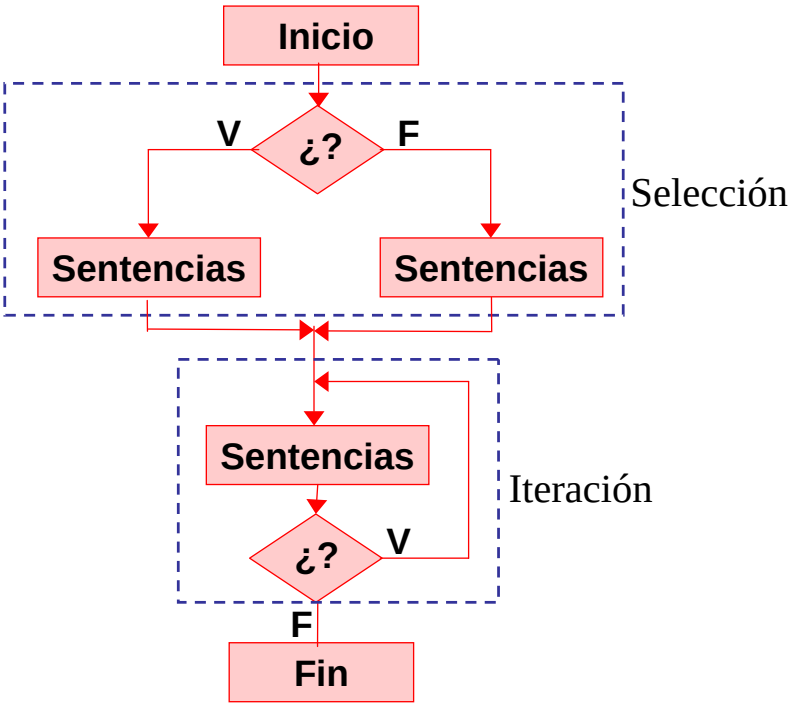
ESTRUCTURA SECUENCIAL

- Estructura Secuencial: aquella en la que las instrucciones o sentencias son ejecutadas una a continuación de la otra en un determinado orden
- Se puede alterar esa secuencialidad usando estructuras o sentencias de control. Estas estructuras permiten variar el flujo de control del programa dependiendo de ciertas condiciones
  - Estructuras de Selección: Permiten que se tomen rutas alternativas de acción dependiendo del resultado de una condición
  - Estructuras de Iteración: **(de repetición o bucle)**: Permiten repetir un conjunto de sentencias un número determinado de veces

ESTRUCTURA SECUENCIAL



Estructura Secuencial



Estructura no Secuencial

Tema 4: Estructuras de Control
Estructura y Contenidos

ESTRUCTURAS DE SELECCIÓN

- Controlan la selección de flujos alternativos en un algoritmo
- Permiten seleccionar una sentencia o grupo de sentencias en función de una condición que, normalmente, es una expresión lógica
- Tipos de sentencias de selección:
  - Selección Simple → **`if`**
  - Selección Binaria → **`if ... else`**
  - Selección Múltiple → **`switch`**

EXPRESIONES LÓGICAS

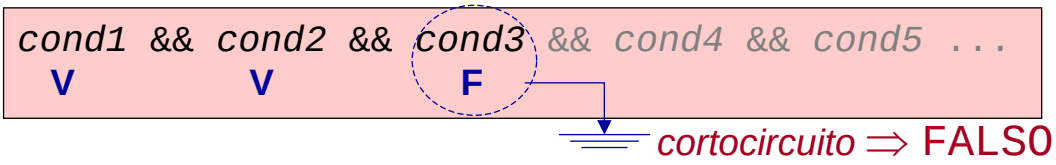
- Las expresiones lógicas son aquellas en las que aparecen exclusivamente operadores relacionales y/o lógicos (`>`, `<`, `>=`, `<=`, `==`, `!=`, `&&`, `||`, `!`) y paréntesis (en caso necesario)
- El resultado de una expresión lógica es verdadero (`≠0`) o falso (`0`)

```
es_primo
x >= 0
!((letra=='Q') || (letra=='z')) && (x>=6)
(largo==ancho) && !(x<5)
x && (altura<=5)
(m!=10) || (z>5)
```

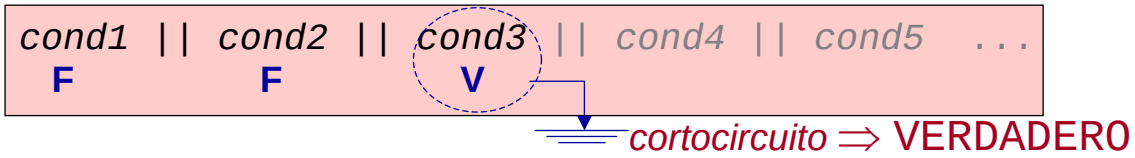
- Las expresiones lógicas se utilizan en las estructuras de control para permitir alterar el flujo del programa

EXPRESIONES LÓGICAS

- En el lenguaje C, como en otros lenguajes de programación, las expresiones lógicas se evalúan en cortocircuito: al evaluar un operando podría concluirse que toda la expresión lógica fuese verdadera o falsa
  - Cuando un operando de una conjunción lógica (&&) es falso, toda la expresión es falsa



  - Cuando un operando de una disyunción lógica (||) es verdadero, toda la expresión lógica es verdadera



EXPRESIONES LÓGICAS

- Leyes de Morgan:

```
!(cond1 && cond2 && cond3 && ... )
                ⇔
  !cond1 || !cond2 || !cond3 ...
```

```
!(cond1 || cond2 || cond3 || ... )
                ⇔
  !cond1 && !cond2 && !cond3 ...
```

```
!(b>=a && sqrt(x)>=1.0)
          ⇔
  (b<a) || sqrt(x)<1.0
```

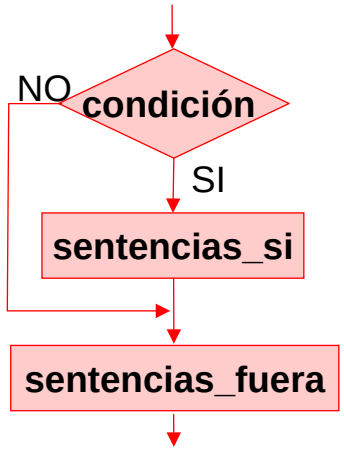
```
!(a==b || ok || !(x<y))
          ⇔
  a!=b && !ok && (x<y)
```

EXPRESIONES LÓGICAS

- Una de las características más importantes del C (y en ocasiones difíciles de interpretar) es su flexibilidad para combinar expresiones y operadores de distintos tipos
- Se acaba de mostrar que el resultado de una expresión lógica es siempre un valor numérico (`0` ó `≠0`). Esto permite que cualquier expresión aritmética pueda aparecer como sub-expresión en una expresión lógica
- A su vez, el operador de asignación (`=`), además de asignar un valor a una variable, deja este valor disponible. Por tanto, una expresión de asignación puede también aparecer como sub-expresión en una expresión lógica

```
!(a>3) && (b=c=d) || (8-f++)
```

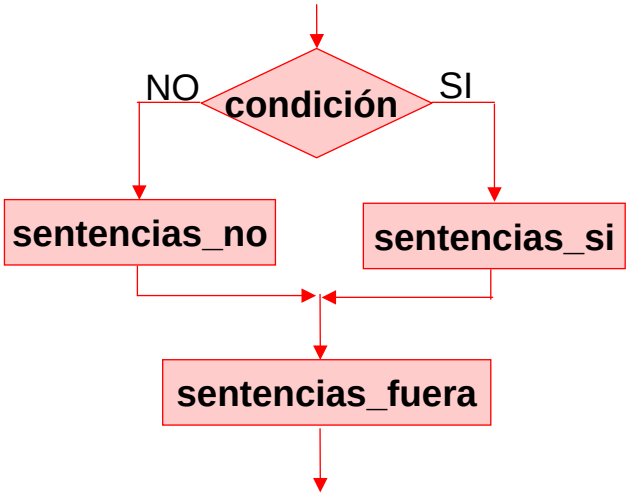

SELECCIÓN SIMPLE




```
if (<condición>)
  {
     <sentencias_si>
  }
<sentencias_fuera>
```

- `<condición>` es una expresión que puede ser verdadera o falsa
  - Recordemos que en Ansi C no existen tipos lógicos, verdadero es cualquier valor distinto de cero y falso es el valor cero
  - Por tanto `<condición>` puede ser cualquier expresión
- `<condición>` tiene que ir SIEMPRE entre paréntesis
- Si hay una única `<sentencia_si>` se pueden omitir las llaves

SELECCIÓN BINARIA




```
if (<condición>)
  {
     <sentencias_si>
  }
else
  {
     <sentencias_no>
  }
<sentencias_fuera>
```

- Las mismas consideraciones que para la selección simple

SELECCIÓN SIMPLE Y BINARIA

```c
/* Determina el número más grande entre tres */
#include <stdio.h>
int main()
{
  int n1, n2, n3, mayor;
  printf("Primer valor: ");   scanf("%d",&n1);
  printf("Segundo valor: ");  scanf("%d",&n2);
  printf("Tercer valor: " );  scanf("%d",&n3);
  if (n1 > n2) /* Calcular el mayor de n1 y n2 */
     mayor = n1;
  else
     mayor = n2;
  if (n3 > mayor) /* Ver si n3 es el mayor */
     mayor = n3;
  printf("El mayor de %d,%d,%d es %d\n",n1,n2,n3,mayor);
  return 0;
}
```

SELECCIÓN MÚTIPLE

```
switch ( <selector> )
{
  case <valor_1>:<Sentencias_1>
                  break;
  case <valor_2>:<Sentencias_2>
                 break;
  case <valor_3>:<Sentencias_3>
                 break;
  default:      <Sentencias_df>
}
```

¿Cómo funciona la sentencia `switch`?

1. Se evalúa el selector
2. Comparación del selector con los `<valores_i>`
3. Ejecución de las `<sentencias_i>` correspondientes
4. Cada caso termina con un `break`, o se continua con `<sentencias_i+1>`

- `<selector>` tiene que ir SIEMPRE entre paréntesis
- `<selector>` tiene que ser de tipo ordinal
- Si el selector no coincide con ningún `<valor_i>` se ejecutan las `<sentencias_df>`
- `default` no es obligatorio

SELECCIÓN MÚTIPLE

```
/* Ejemplo de un menú muy simple */
#include <stdio.h>
int main()
 {
   int opcion;
   printf("1.España\n  2.Francia\n  3.Italia\n");
   printf("4.Inglaterra\n");
   printf("Selecciona una opción: ");
   scanf("%d", &opcion);
   switch (opcion)
    {
      case 1:  printf("Hola\n");   break;
      case 2:  printf("Allo\n");   break;
      case 3:  printf("Pronto\n"); break;
      case 4:  printf("Hello\n");  break;
    }
   return 0;
 }
```

SELECCIÓN MÚTIPLE

```
/* Ejemplo de cómo englobar varios casos */
#include <stdio.h>
int main()
{
   char letra;
   printf("Introduce una letra: ");
   scanf("%c",&letra);
   switch (letra) /* Aquí el selector es una variable */
    { case 'a':   /* de tipo carácter*/
      case 'e':
      case 'i':
      case 'o':
      case 'u': printf("Es una vocal minúscula\n"); break;
      case 'A': case 'E': case 'I': case 'O':
      case 'U': printf("Es una vocal mayúscula\n");
                break;
      default:  printf("No es vocal\n");
    }
   return 0;
}
```

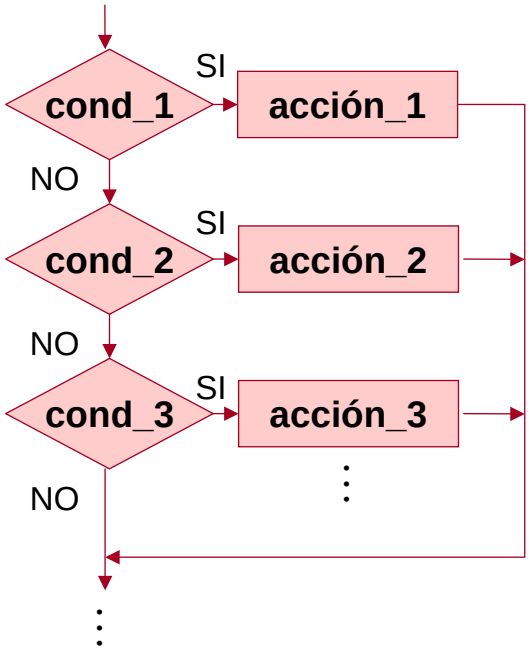

ANIDAMIENTOS

- Una sentencia de selección puede contener otra en cualquier rama

```
/*trozo de código con anidamientos */
if (n1>n2)
 {  /*Estas llaves se pueden omitir*/
    if (n1>n3)
         printf("El máximo es ", n1);
    else
         printf("El máximo es ", n3);
 }
else
 {
    if (n2>n3)
         printf("El máximo es ", n2);
    else
         printf("El máximo es ", n3);
 }
```

ANIDAMIENTOS

Selección Múltiple



ANIDAMIENTOS

```
/*caso binario especial*/
if (cond_1)
      acción_1;
else
      if (cond_2)
            acción_2;
      else
            if (cond_3)
                acción_3;
...
```

```
/* sintaxis abreviada */
if (cond_1)
      acción_1;
else if (cond_2)
      acción_2;
else if (cond_3)
      acción_3;
...
```

¡Importante la indentación!

```
if (nota<5)
      printf("Suspenso");
else if (nota<7)
      printf("Aprobado");
else if (nota<9)
      printf("Notable");
else
      printf("Sobresaliente");
```

ANIDAMIENTOS

```
switch (operador)
 {
  case '+' : resultado  = a+b;
             break;
  case '-' : resultado  = a-b;
             break;
  case '*' : resultado  = a*b;
             break;
  case '/' : resultado  = a/b;
             break;
  default:printf("\nIndefinido");
 }
```

Es más legible que...

```
if (operador == '+')
  resultado = a+b;
else
  if (operador == '-')
    resultado = a-b;
  else
    if (operador == '*')
      resultado = a*b;
    else
      if (operador == '/')
        resultado = a/b;
      else
        printf("\nIndefinido");
```


ESTRUCTURAS DE SELECCIÓN

Desarrolla un programa que lea los coeficientes de una ecuación de segundo grado y calcule e imprima sus soluciones.

Deben contemplarse todos los posibles casos que se puedan dar.

Diseña primero el algoritmo mediante diagramas de flujo

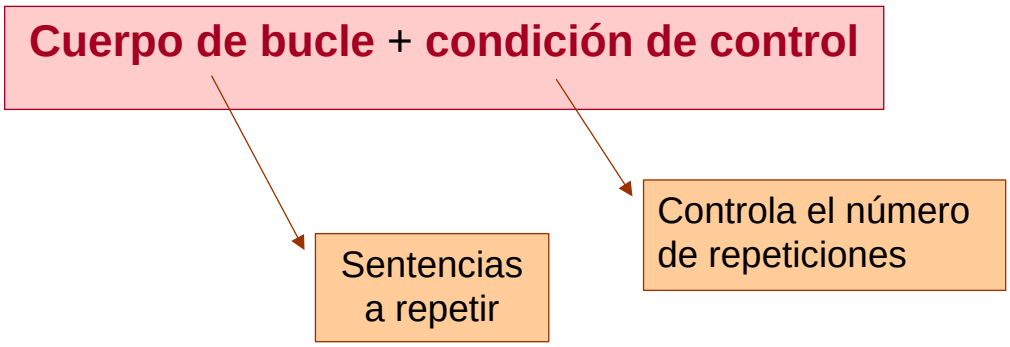
Desarrolla un programa que lea tres números naturales, correspondientes al día, mes y año de una fecha, e indique si es una fecha válida del siglo XX

Desarrolla un programa que lea cinco letras que corresponden a una hora en formato 24h (hh:mm) e imprima la hora en formato 12h acompañada de AM o PM según proceda (p.e. 09:35 -> 09:35 AM ; 18:15 -> 06:35 PM)

Tema 4: Estructuras de Control
Estructura y Contenidos

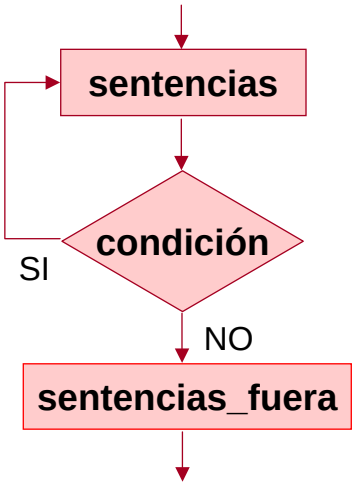
ESTRUCTURAS DE REPETICIÓN

- También llamadas estructuras de iteración o bucles
- Permiten repetir una sentencia o grupo de sentencias un número o determinado o indeterminado de veces a priori: bucles deterministas y bucles no deterministas (post-condición y pre-condición)
- Una condición de control (expresión lógica) permite controlar el número de repeticiones



BUCLES NO DETERMINISTAS

- Bucle post-condición: **`do...while`**
  - la condición de terminación se evalúa después de cada iteración




```
do{
     <sentencias>
}while(<condición>);
<sentencias_fuera>
```

`<condición>` tiene que ir SIEMPRE entre paréntesis

`<sentencias>` se ejecuta 1 ó más veces

BUCLES NO DETERMINISTAS

- Bucle post-condición: **`do...while`**

```
printf("Hola\n");
contador = 1;
do{
  printf("-");
  contador++;
while(contador <= 4);
```

4
contador

```
Hola
----
```

BUCLES NO DETERMINISTAS

- Ejemplo: cálculo aproximado de e^x

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

```
scanf("%f",&x);
cont=0; serie=0.0;
fact=1.0; pot=1.0;
do{
  serie = serie + pot / fact;
  cont++;
  fact = fact * (float)cont;
  pot = pot * x;
}while(pot/fact > 0.000001 );
```

Hasta aportación de cada término menor que 0.000001

  - No se sabe de antemano las veces que se itera el bucle
  - Vamos a mostrar la ejecución de tres iteraciones

BUCLES NO DETERMINISTAS

- Ejemplo: validación de entradas por teclado

```
do{
  printf("Número de mes:" )
  scanf("%d",&mes);
}while (!( mes>=1 && mes<=12 ));
do{
  printf("Día del mes:" )
  scanf("%d",&dia);
}while (!( dia>=1 && dia<=31 ));
```

Validar número de mes

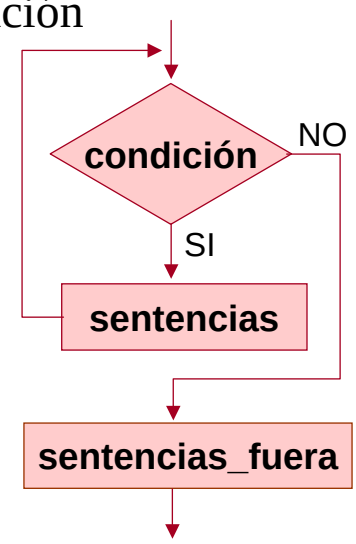
Validar día de mes

BUCLES NO DETERMINISTAS

```c
/* Calcula el número positivo más grande de una lista. La
entrada se realiza mientras los números sean mayores que
cero */
#include <stdio.h>
int main()
{
   int num, max=0;
   do{
      printf("Introduce un número: " );
      scanf("%d",&num);
      if( num > max )
         max = num;
   }while( num > 0 );
   if( max!=0 )
      printf("El número más grande es %d.\n", max);
   else
      printf("No se han introducido números.\n");
   return 0;
}
```


BUCLES NO DETERMINISTAS

- Bucle pre-condición: **`while`**
  - la condición de terminación se evalúa antes de cada iteración



cuidado con poner involuntariamente un ;

```
while(<condición>);
 {
    <sentencias>
 }
<sentencias_fuera>
```

`<condición>` tiene que ir SIEMPRE entre paréntesis

`<sentencias>` se ejecuta 0 ó más veces

BUCLES NO DETERMINISTAS

- Bucle pre-condición: **`while`**
  - Algunos ejemplos muy utilizados son:
    - Bucle controlado por contador
    - Bucle controlado por centinela
    - Bucle contador

BUCLES NO DETERMINISTAS

- Bucle controlado por contador
  - Se ejecuta un numero determinado de veces
  - Se utiliza una variable de control del bucle (vcb)
  - Componentes
    - Inicialización
    - Comprobación de la condición
    - Actualización

```
i=0;            /*Inicialización*/
while(i<=10)    /*comprobación*/
 {
     <sentencias>
     i++;        /*actualización*/
 }
```

En este tipo de bucles usaremos la sentencia **`for`**

BUCLES NO DETERMINISTAS

- Bucle controlado por centinela
  - centinela → valor especial de una variable que controla el final del bucle
  - Es necesario actualizar el centinela en cada iteración
  - La primera evaluación de la condición exige una actualización adelantada de la variable de control del bucle

```
scanf("%d",&centinela);/*actualización
                                adelantada*/
while(centinela!=0)
 {
   <sentencias>
   scanf("%d",&centinela); /*actualización*/
 }
```


BUCLES NO DETERMINISTAS

- Bucle contador
  - Útil cuando se quiere contar el número de veces que se ejecuta el bucle
  - La condición de terminación no depende del contador

```
/*hace eco por pantalla de la entrada y cuenta
los caracteres hasta encontrarse un punto*/
contador=0;
scanf("%c",&ch);
while(ch!='.')
 {
   putchar(ch);
   contador++;
   scanf("%c",&ch);
 }
printf("Caracteres leidos: %d",contador);
```

BUCLES NO DETERMINISTAS

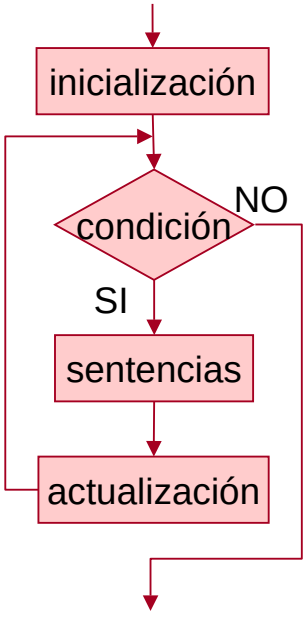
```c
/*Ejemplo de bucle contador: calculo de la media de una
lista de números hasta introducir el cero */
#include <stdio.h>
int main()
 {
   int i=0;
   float  x, suma=0.0;
   printf("Introduce un número: ");
   scanf("%f", &x);
   while ( x != 0 )
    {
      suma = suma + x;
      i++;
      printf("Introduce un número: ");
      scanf("%f", &x);
    }
   if ( i != 0 )
       printf("La media es %f.\n",  suma/(float)i );
   else
       printf("No se han introducido números\n");
   return 0;
 }
```


BUCLES DETERMINISTAS

- El bucle **`for`**
  - Es muy versátil en el lenguaje de programación C

```
for (<inicialización>;<condición>;<actualización>)
{
   <sentencias>
}
```

expresiones



En C la asignación se considera un operador, y, por tanto, las asignaciones son expresiones.

BUCLES DETERMINISTAS

- El bucle **`for`**
  - Lo usaremos cuando el número de veces que se repite el bucle se puede determinar a priori
  - Tendremos siempre una variable de control del bucle (`vcb`)
  - En el cuerpo del bucle, `vcb` puede usarse, NUNCA cambiarse

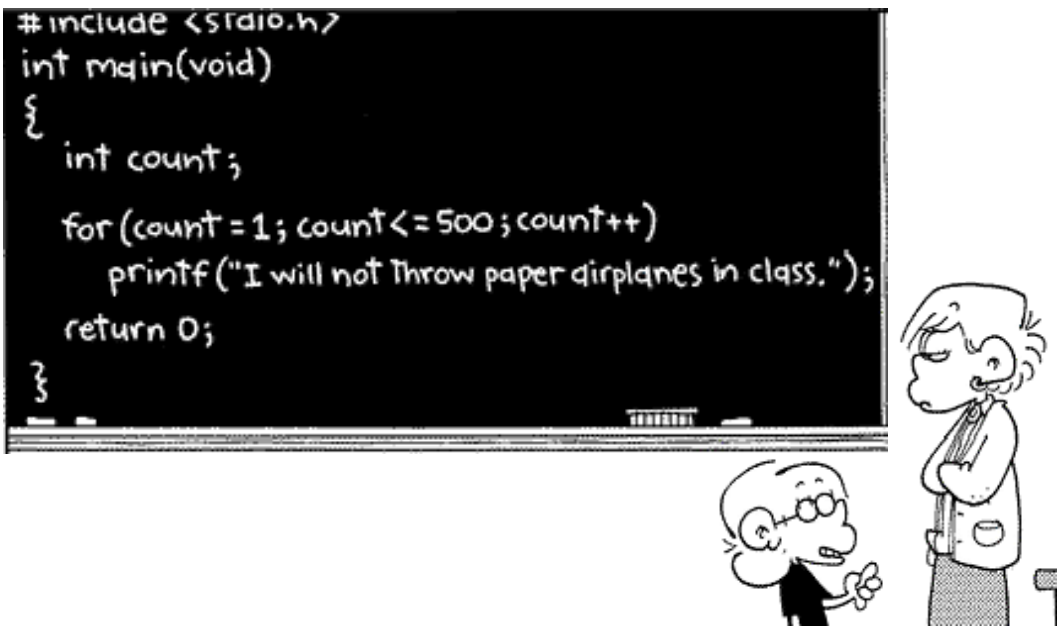
```
for(i=1;i<10;i++)  /* i empieza tomando el valor 1, cada vez que el bucle
    da una iteración se incrementa en 1. El bucle terminará cuando  i  sea igual a 10, es
    decir, el último valor que  i  tomará dentro del bucle será 9 */

for(i=2;i<=128;i*=2)  /* i empieza tomando el valor 2; cada vez que el
    bucle da una iteración i se multiplica por 2. Esto continua mientras i sea menor o
    igual que 128. Los valores de i dentro del bucle serán 2,4,8,16,32,64 y 128  */

for(j=10;j>0;j--)  /* j  empieza tomando el valor 10, cada vez que el bucle
    da una iteración se decrementa en 1. El bucle terminará cuando  j  sea igual a 0, es
    decir, el último valor que  i  tomará dentro del bucle será 1  */
```


BUCLES DETERMINISTAS

- El bucle **`for`** : ejemplo



BUCLES DETERMINISTAS

- El bucle **`for`** : ejemplo

```
/* Tabla de multiplicar */
#include <stdio.h>
int main()
 {
   int i, num;
   printf("Introduce un número :");
   scanf("%d",&num);
   for(i=1; i<=10; i++)
     printf("\n%d x %d = %d",num,i,num*i);
   return 0;
 }
```

```
Introduce numero: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```


BUCLES DETERMINISTAS

- El bucle **`for`** : ejemplo

```
/* Factorial de un número */
#include <stdio.h>
int main()
 {
   int factorial, i, num;
   printf("Introduce un numero :");
   scanf("%d",&num);
   factorial=1;
   for(i=1; i<=num; i++)
     factorial*= i;
   printf("\nEl factorial es %d",factorial);
   return 0;
 }
```

```
Introduce numero: 6
El factorial es 720
```

BREAK y CONTINUE

- La sentencia `break` se utiliza para forzar la salida de un bucle independientemente de que se cumpla o no la condición de terminación.
- La sentencia `continue` termina la iteración en curso y vuelve a evaluar de nuevo la condición de terminación del bucle.
- En programación estructurada, no es recomendable el uso de este tipo de sentencias y por lo tanto **NO LAS USAREMOS**

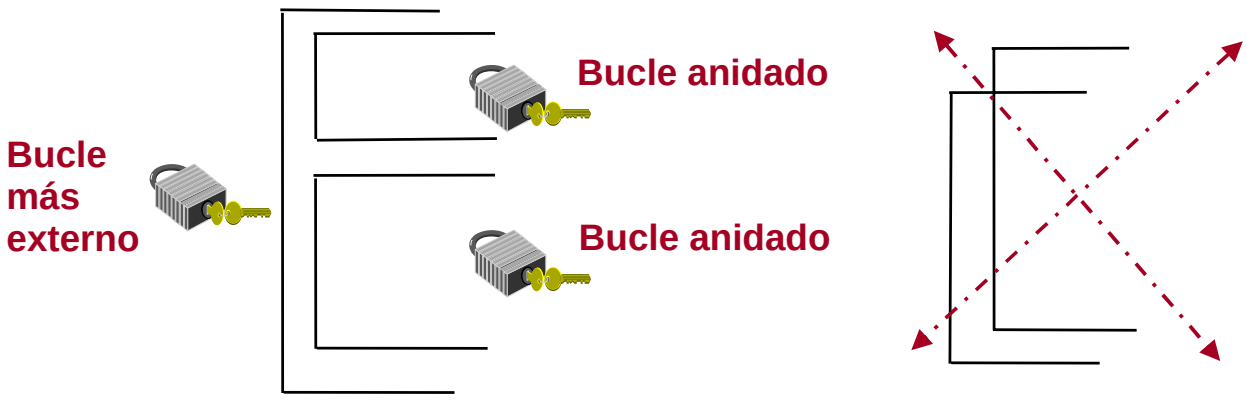
"Se demuestra que todo problema que pueda resolverse en un numero finito de pasos puede expresarse usando únicamente 3 tipos de estructuras o bloques fundamentales, con una sola entrada y una sola salida para organizar dichos pasos:

- Un proceso secuencial.
- Un mecanismo de decisión binaria.
- Un mecanismo de bucle generalizado.“

(Bohm y Jacopini, 1965)

ANIDAMIENTOS DE BUCLES

- Al igual que en las estructuras selectivas, no hay restricciones en las sentencias del cuerpo del bucle
- La estructura interna debe de estar totalmente incluida en la externa



ANIDAMIENTOS DE BUCLES

```
/*Escribe un rectángulo de asteriscos*/
#include <stdio.h>
int main()
 {
   int i, j;
   for(i=1;i<=3;i++)
    {
      for(j=1;j<=8;j++)
        printf("*");
      printf("\n");
    }
   return 0;
 }
```

bucle exterior

bucle interior

```
********
********
********
```


ANIDAMIENTOS DE BUCLES

```
/*Escribe un rectángulo de asteriscos*/
#include <stdio.h>
int main()
 {
   int i, j;
   for(i=1;i<=3;i++)
    {
      for(j=1;j<= i ;j++)
        printf("*");
      printf("\n");
    }
   return 0;
 }
```

bucle exterior

bucle interior

```
*
**
***
```

ANIDAMIENTOS DE BUCLES

Basándote en el código anterior, modifica el programa para que obtenga los ejemplos adjuntos. La altura del triángulo se pide por teclado.

```
Introduce altura: 3
  1
 12
123
```

```
Introduce altura: 4
   1
  121
 12321
1234321
```


ANIDAMIENTOS DE BUCLES

```
/* Encuentra el primer número perfecto mayor que 28.
Un número es perfecto si coincide con la suma de sus
Divisores, sin contarse el mismo.    Ej:  6 = 1+2+3 */
#include <stdio.h>
int main()
 {
  int encontrado = 0, intento, cont, suma;
  intento = 29; /* empiezo con el siguiente de 28 */
  while(!encontrado)
    {
     suma=1; /* el 1 es divisor de todos los numeros */
     for (cont=2; cont<intento; cont++ )
       if ((intento%cont)==0) /* si cont es divisor */
           suma+=cont;        /* lo sumo */
     if (suma == intento )
        encontrado = 1; /* si coincide lo hemos encontrado */
     else
        intento++;      /* sino, probamos con el siguiente */
    }
  printf("Número perfecto mayor que 28 = %d",intento);
  return 0;
 }
```

BUCLES INFINITOS

- Un bucle infinito es un bucle que nunca acaba
- Normalmente se produce cuando un bucle no alcanza la condición de finalización

```
suma = 0;
N = 0;
while (N <= 100)
    suma = suma + N;
printf("%d",suma);
```

N++;

```
suma = 0;
N = 0;
while (N <= 100);
 {
   suma = suma + N;
   N++;
 }
printf("%d",suma);
```

- Mucho cuidado con los puntos y coma tras las condiciones
- Por tanto, al diseñar un bucle, hay que comprobar siempre que las condiciones de finalización serán alcanzadas

EJEMPLOS

Desarrolla un programa que lea una secuencia de números naturales terminada en cero e indique la posición de la primera y última ocurrencia del número doce.

Desarrolla un programa que, dado un número natural leído por teclado, indique si es o no número primo. El programa preguntará si se quiere repetir de nuevo el proceso.

Desarrolla un programa que, dada una secuencia de ceros y unos terminada en un punto, muestre por pantalla el tamaño de todas las subcadenas de unos e indique cual es la mayor.

FIN DEL TEMA